Predicting Transport Integrals with Machine Learning

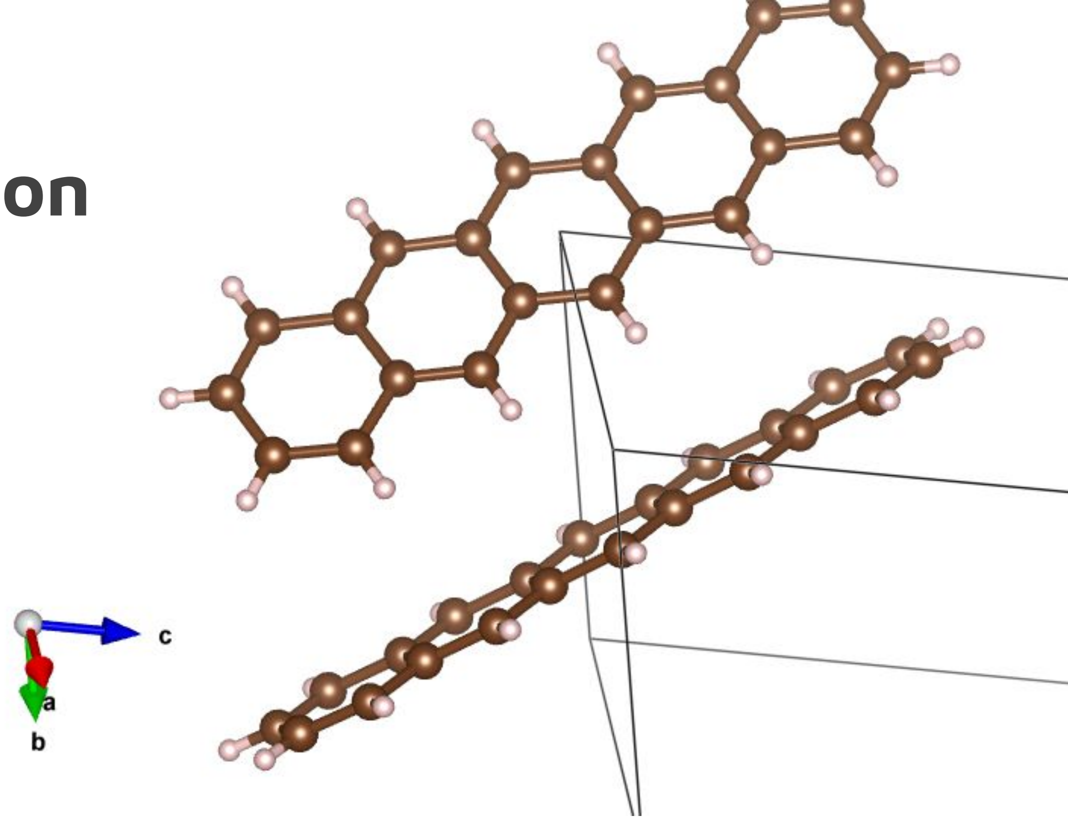
Jeremy Thaller and Maxim Gorbatschow

05.02.2020

Quick Recap

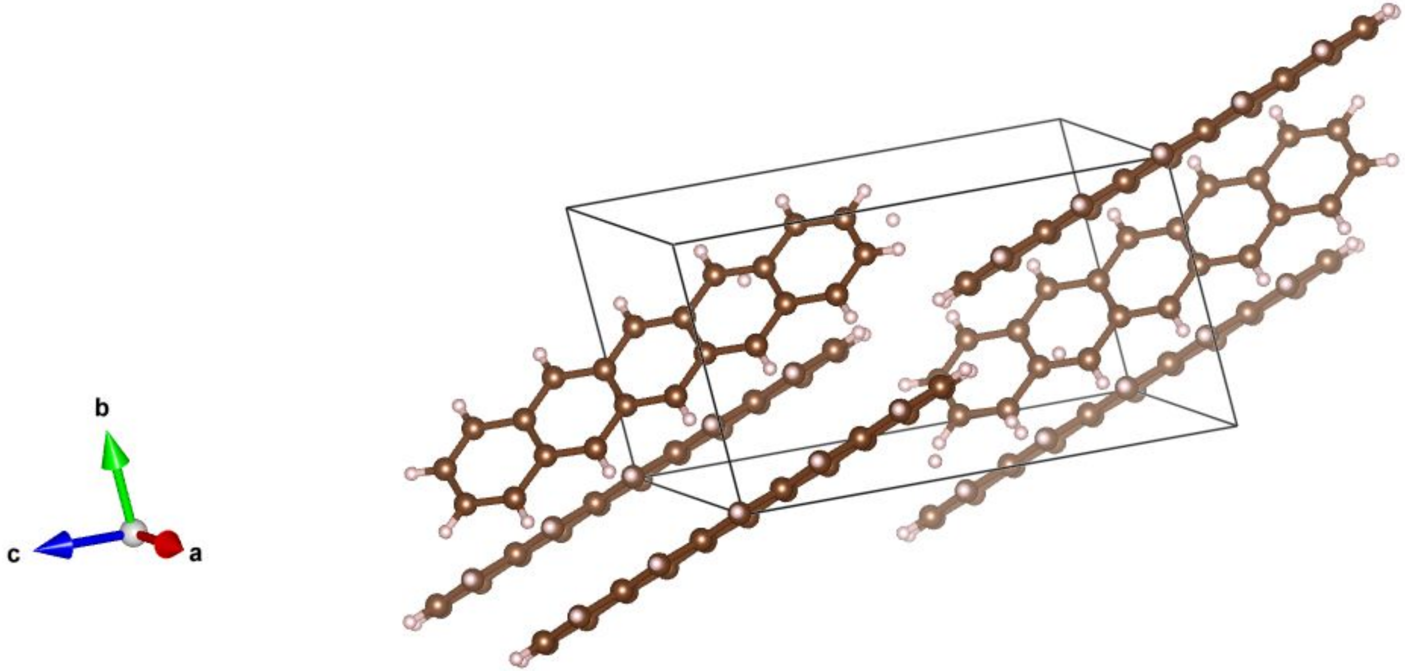
Given Information

Pic file with 900 features and 10,000 example



Goal

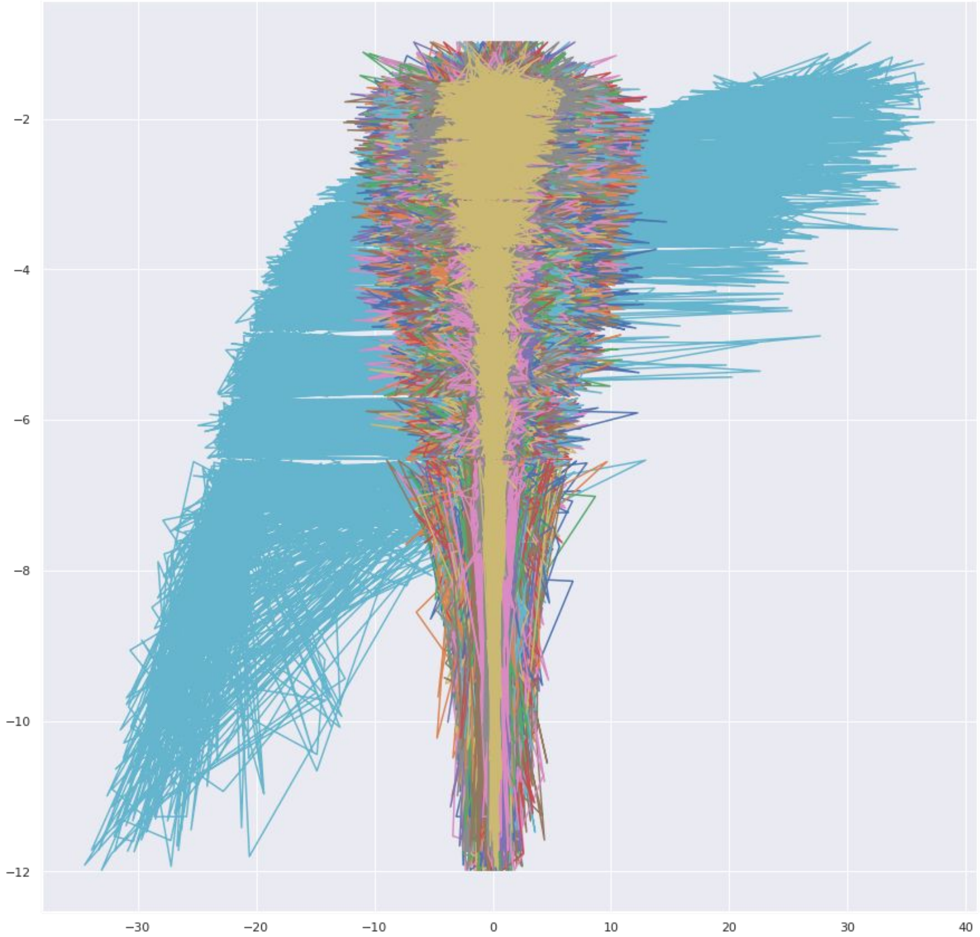
Predict the Transport integral give 900 distances between two 30-atom monomers



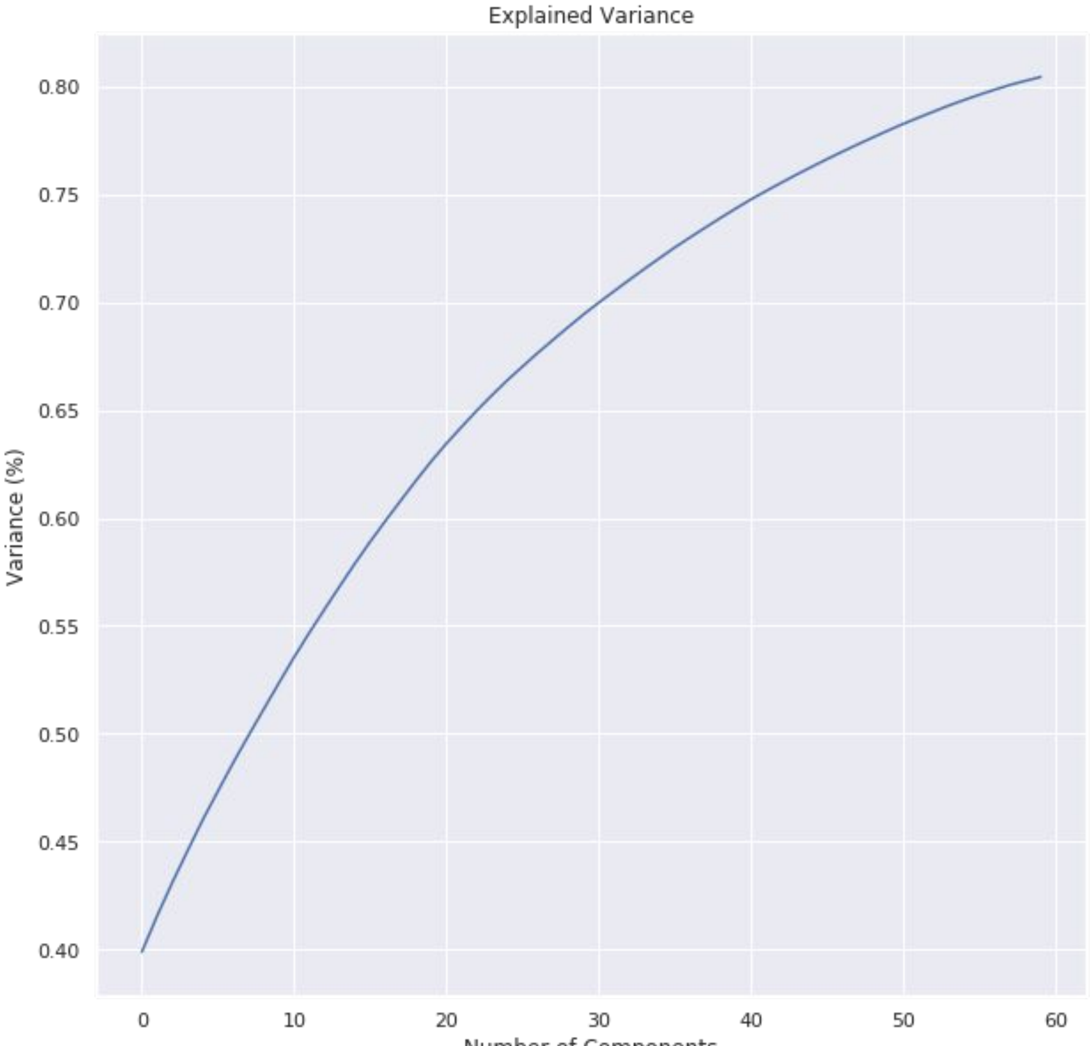
Preprocessing

Dealing with 900 features

Principal Component Analysis



PCA Cont.



Explained Variance
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
Variance (%)
0
10
20
30
40
50
60
Number of Components

Regularization for Model Fitting

Add an extra cost to the loss function to prevent overfitting

L1 vs. L2

Regularization for Model Fitting

L1: Lasso Regression

- Loss function = sum of absolute values of errors
- (**L**east **A**bsolute **S**hrinkage and **S**election **O**perator)
- Tends to push weights to zero

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

L2: Ridge Regression

- Loss function = sum of squared errors

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \boxed{\lambda \sum_{j=1}^{n} \theta_j^2}\right]$$

$$\min_{\theta} J(\theta)$$

Lasso shrinks the less important features' coefficients (even removing them altogether), making it better for feature selection with large number of features

Regression Models

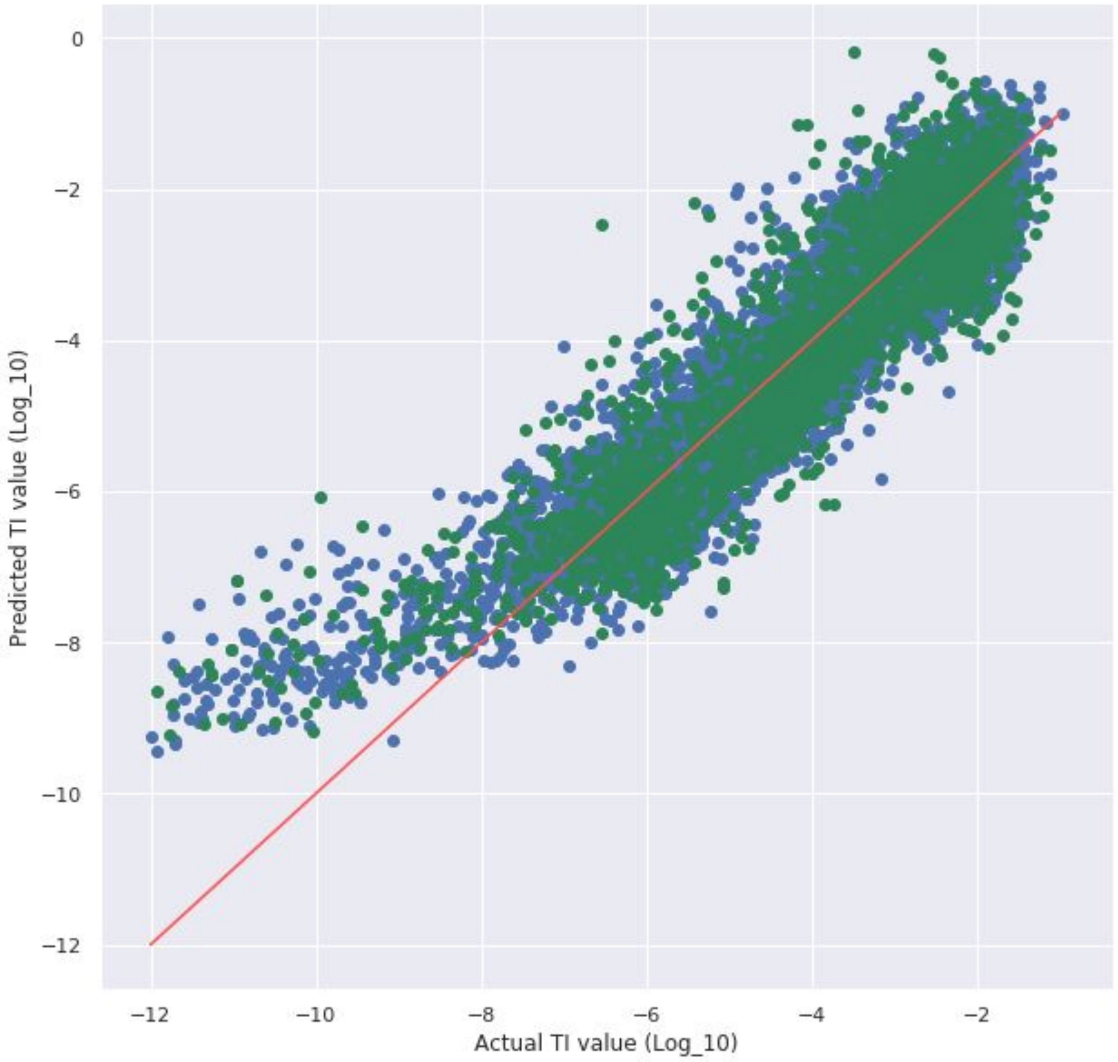
Linear Regression

With PCA applied and k-fold cross validation:

Mean_squared_error (average):
1.002949261587887
standard_deviation:
1.187689627016626

(RMSE: 1.0014735)

Training Data: Y vs. Y_prediction
Test Data: Y vs. Y_prediction
(Linear with slope 1 => perfect match)



Sanity check: How is PCA doing?

Without PCA

```
RMSE:
0.795003 (no k-fold)

mean_squard_error:
1.002949261587887

standard_deviation:
1.187689627016626
```

With PCA

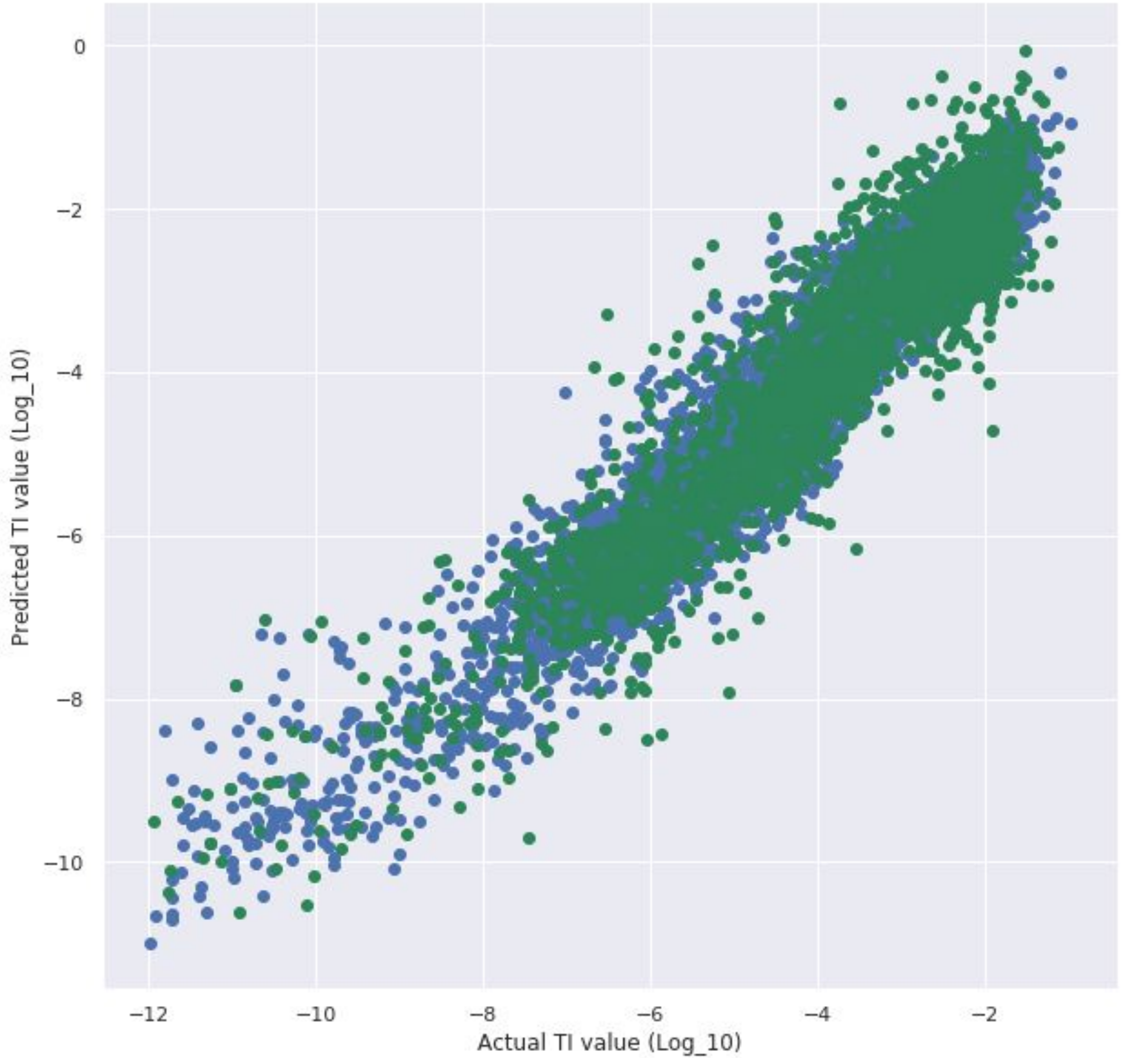
```
RMSE:
1.129056 (no k-fold)

mean_squard_error:
1.002949261587887

standard_deviation:
1.187689627016626
```

Polynomial Regression

Mean_squared_error (average):
1.774617823490909

standard_deviation:
2.4438984231990983

(RMSE: 1.3321478234381157)



Training Data: Y vs. Y_prediction
Test Data: Y vs. Y_prediction

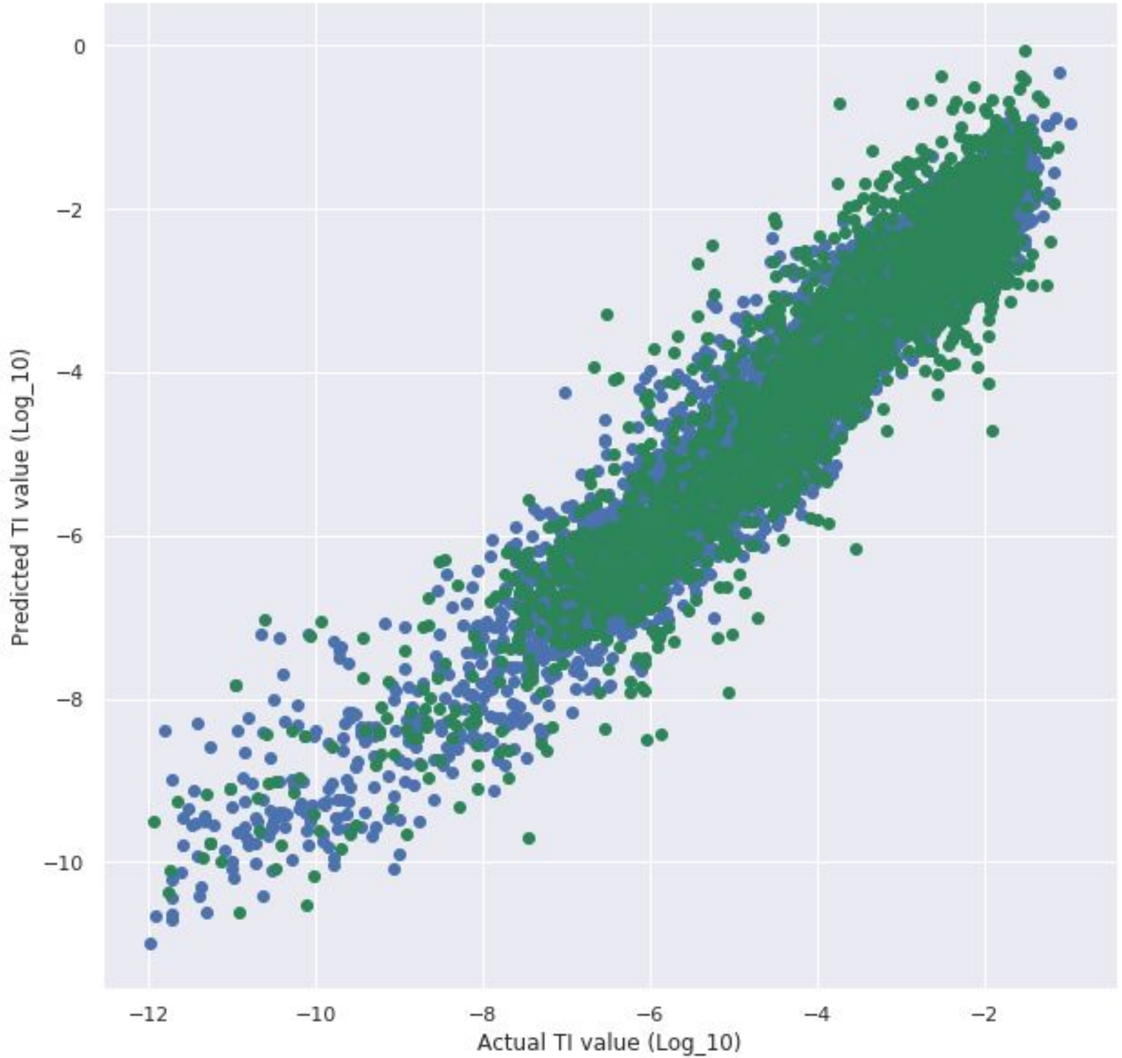
Polynomial Regression

Mean_squared_error (average):

1.774617823490909

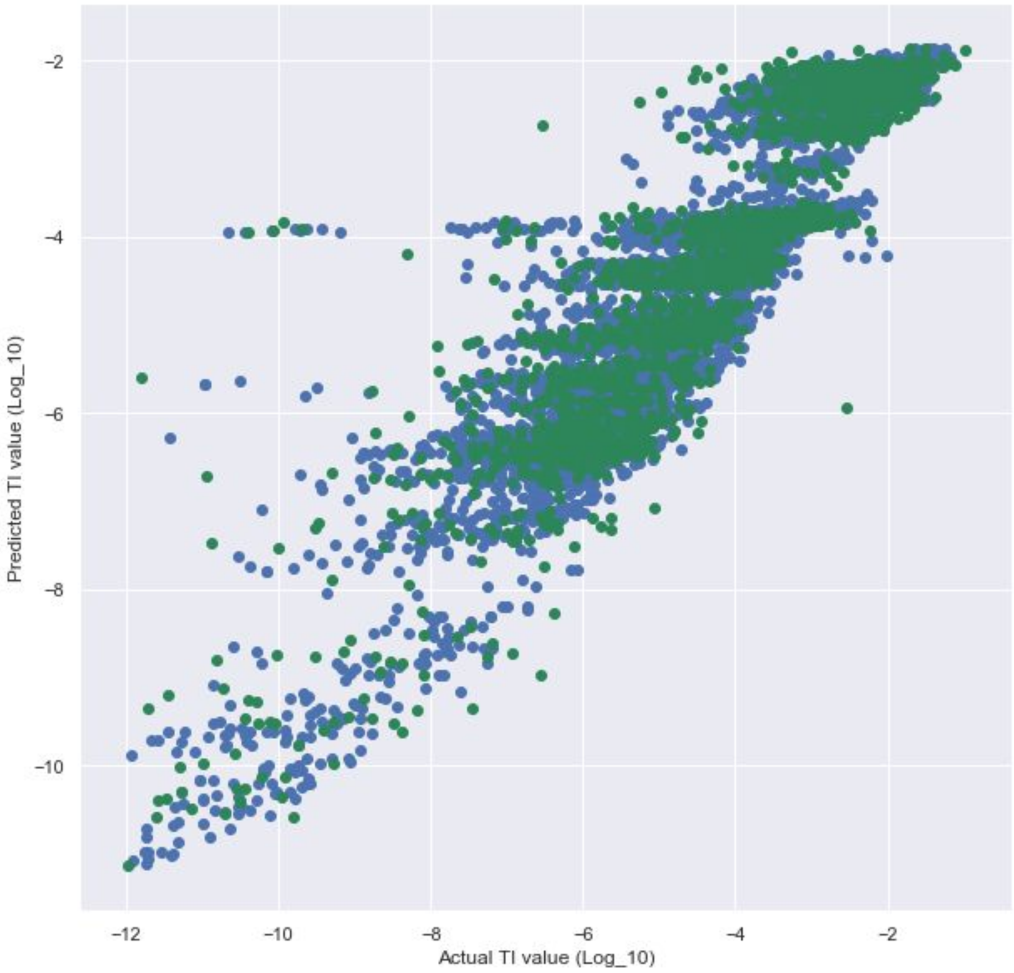
mean_squared_errors: [8.92539671
1.24277882 0.53544367 0.97998094
1.85069762 1.98805718 0.88201865
0.455358   0.37556103 0.51088562]

Training Data: Y vs. Y_prediction
Test Data: Y vs. Y_prediction



Random Forest Regressor

- best parameters are max_depth of 6 and n_estimators of 1000
- RMSE: 0.699682

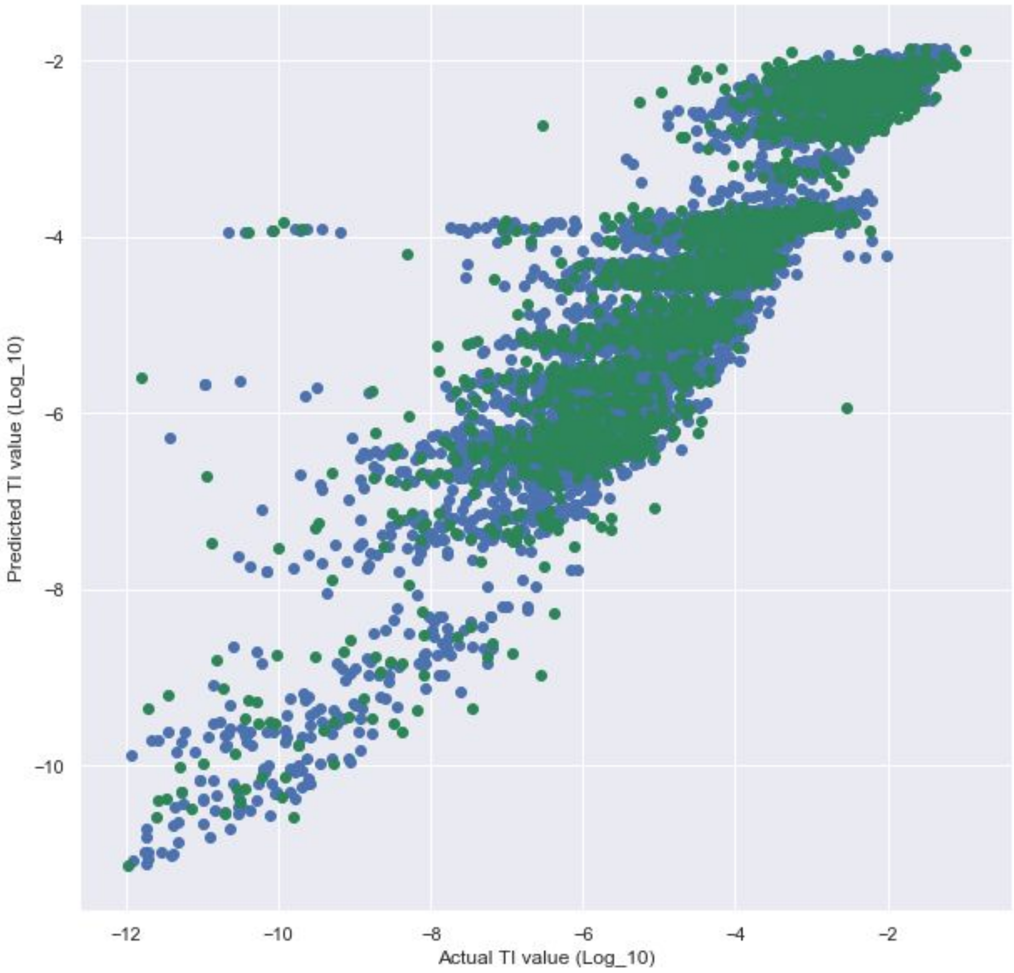


Random Forest Regressor

[6.35710053 0.71129973
0.79274011 0.58765779
0.48054221 0.75907597
0.38439998 0.17103299
0.18910843 0.59895555]

mean_squared_error:
1.1031913291487907

standard_deviation:
1.763541449709036



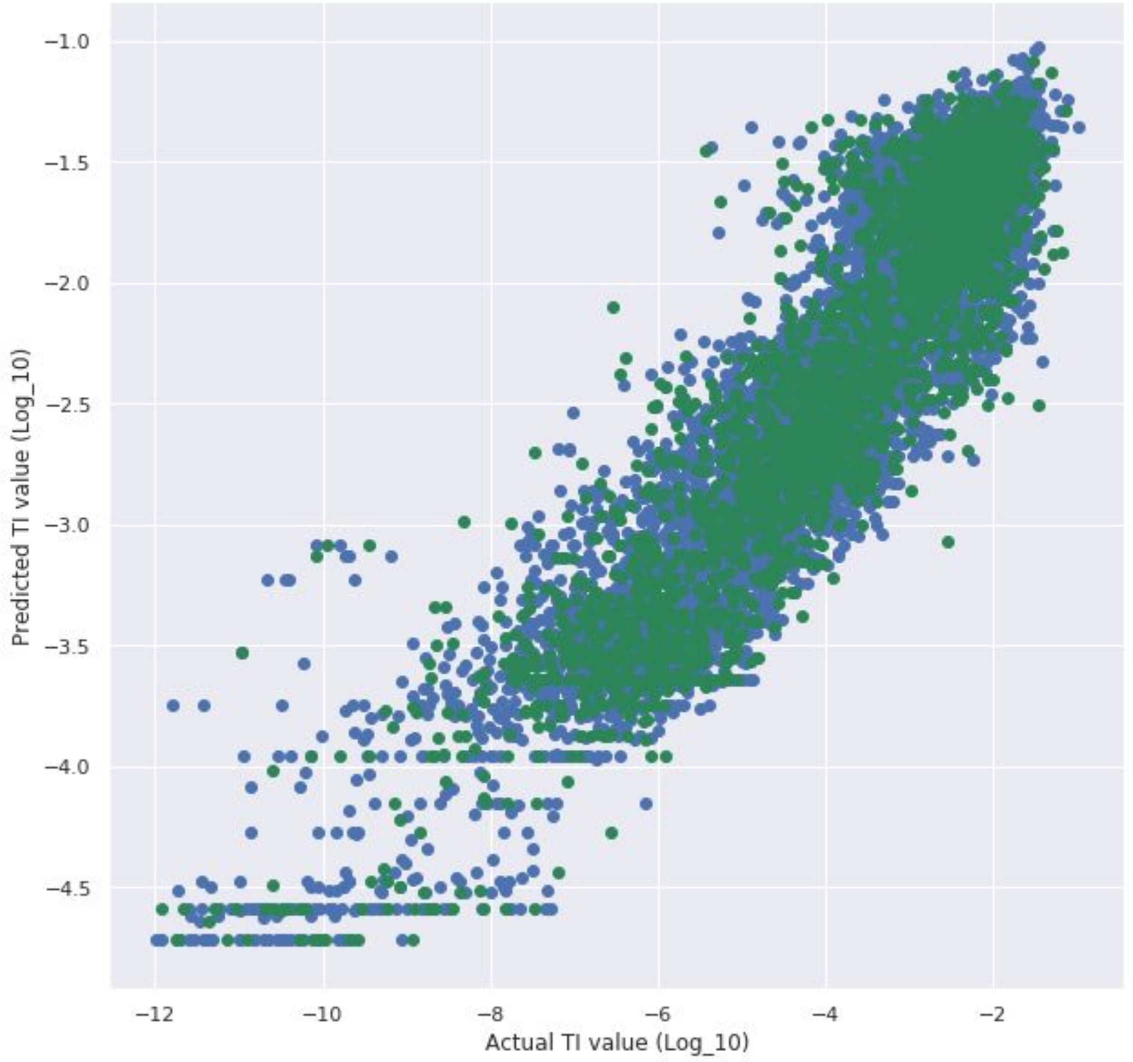
Training Data: Y vs. Y_prediction
Test Data: Y vs. Y_prediction

XGBoost Regressor

RMSE: 1.999990

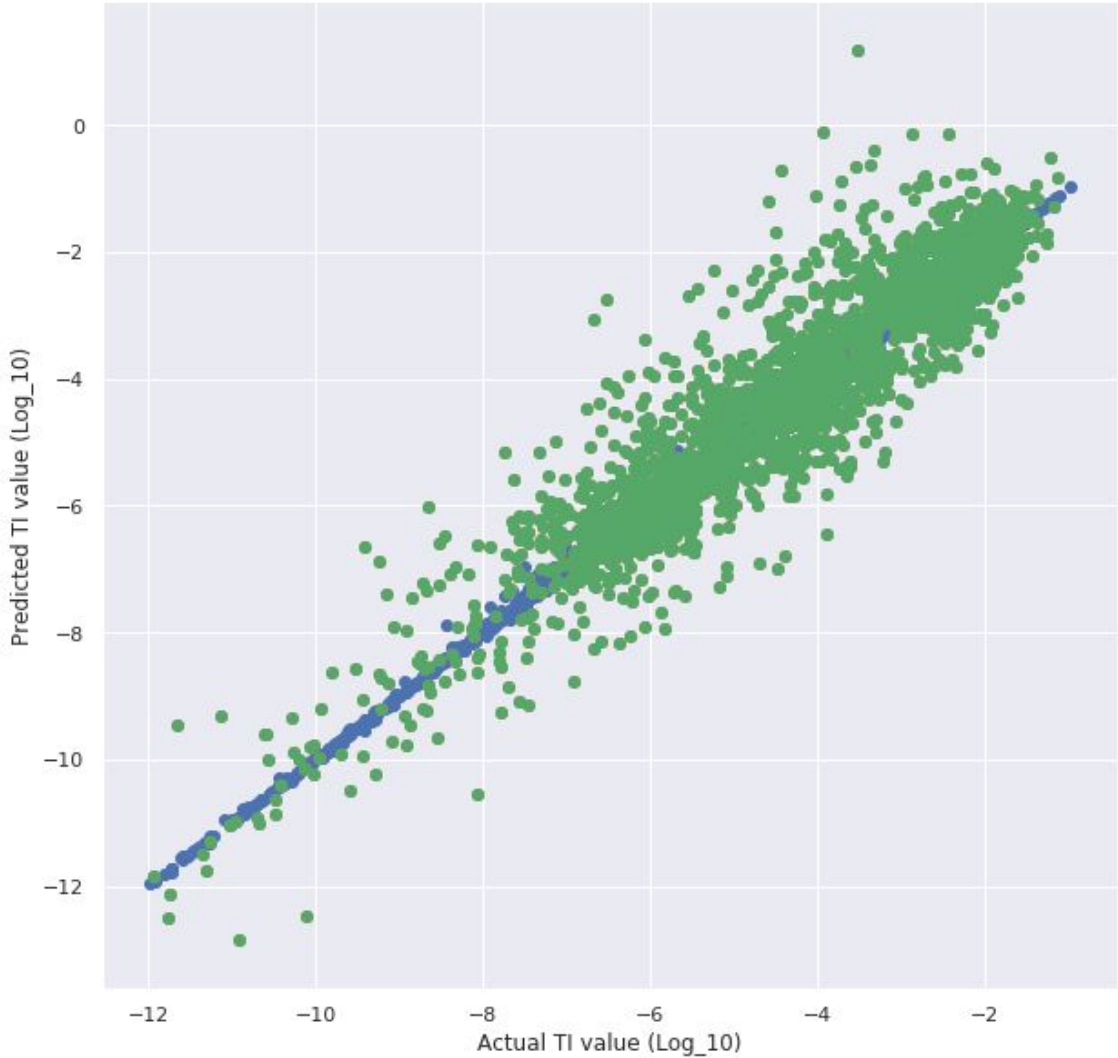
```
xgboost =
xgb.XGBRegressor(objective
='reg:squarederror',
colsample_bytree = 0.3,
learning_rate = 0.1, max_depth =
5, alpha = 10, n_estimators =10)
```

Training Data: Y vs. Y_prediction
Test Data: Y vs. Y_prediction



Kernel Ridge

RMSE of 0.799

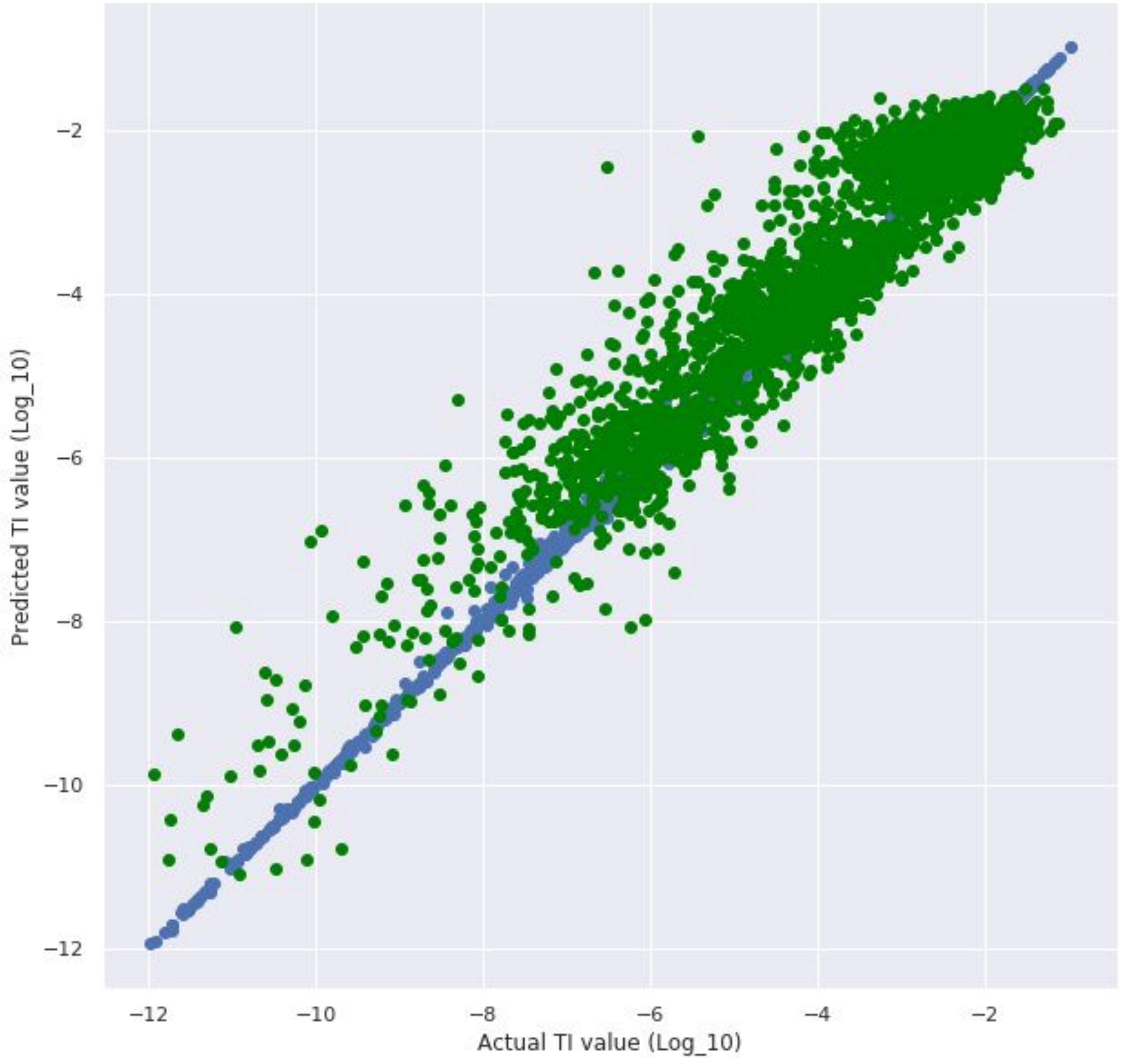


Training Data: Y vs. Y_prediction
Test Data: Y vs. Y_prediction

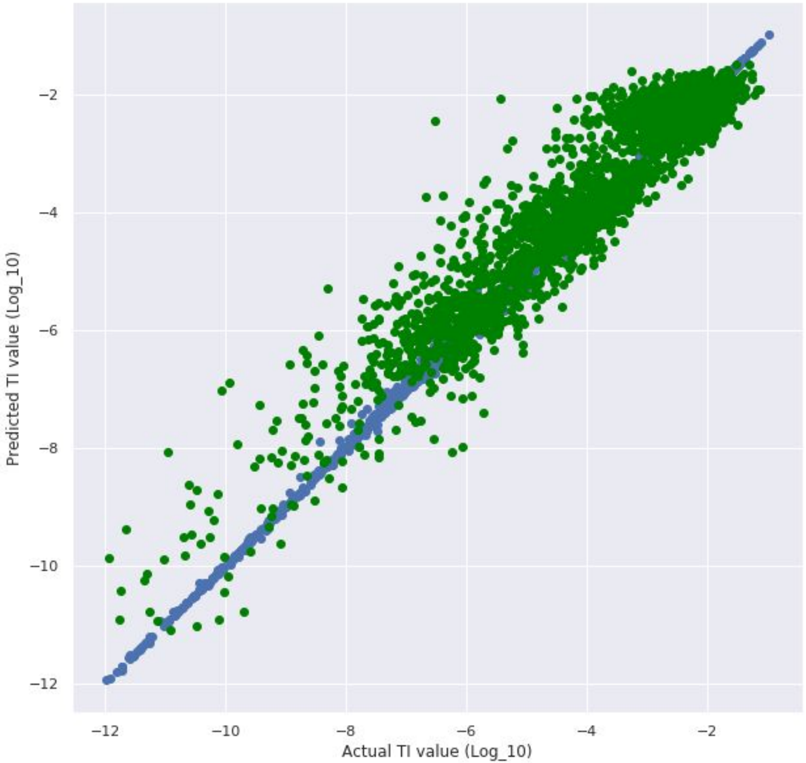
Keras Neural Network

RMSE: 0.645438

Training Data: Y vs. Y_prediction
Test Data: Y vs. Y_prediction

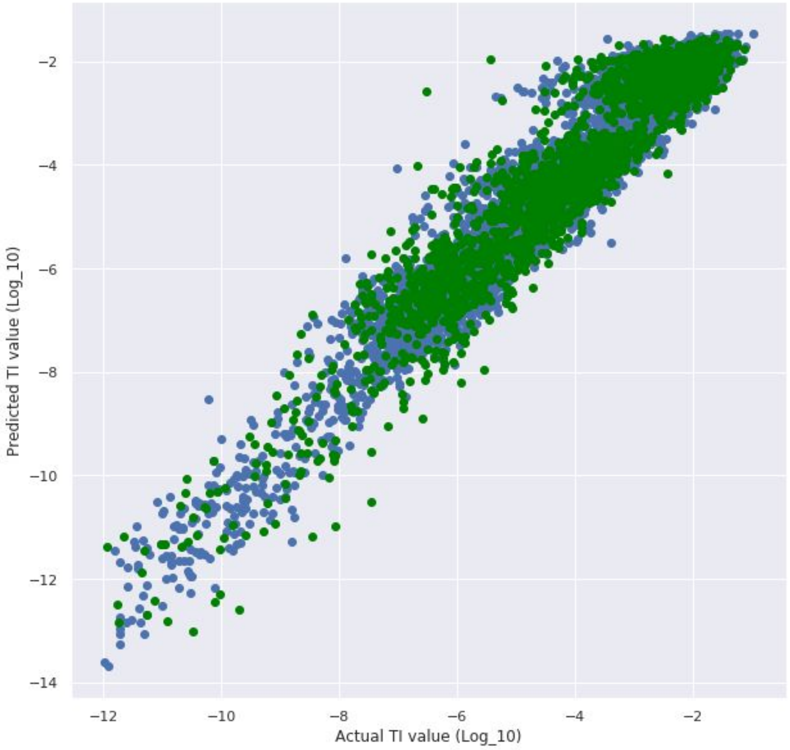


Correcting Keras Overfitting



Loss = mean_absolute_error

RMSE: 0.640988



Loss = mean_squared_error

RMSE: 0.575086

Final Predictions

Keras

```
#fit keras
neuraln = Sequential()
neuraln.add(Dense(128, input_dim=60, activation='relu'))
neuraln.add(Dense(256, kernel_initializer='normal',activation='relu'))
neuraln.add(Dense(256, kernel_initializer='normal',activation='relu'))
neuraln.add(Dense(256, kernel_initializer='normal',activation='relu'))
neuraln.add(Dense(1, kernel_initializer='normal', activation='linear'))
neuraln.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mean_absolute_error'])
neuraln.fit(x_pca, y, epochs=100, batch_size=10)
y_pred = neuraln.predict(x_test_pca)
y_pred = 10 ** y_pred #undo the y = np.log10(TI_df) used by the training
```